Regulator: Dynamic Analysis to Detect ReDoS

Robert McLaughlin, Fabio Pagani, Noah Spahn,
Christopher Kruegel, Giovanni Vigna

University of California, Santa Barbara

So, About That Code Review …

```
const normalizeDataURL = (urlString, {stripHash}) => {
   const match = /^data:(?<type>.*?),(?<data>.*?)(?:#(?<hash>.*))?$/.exec(urlString);

   if (!match) {
       throw new Error(`Invalid URL: ${urlString}`);
   }

   let {type, data, hash} = match.groups;

   // ...
}
```

So, About That Code Review ...

```
const normalizeDataURL = (urlString, {stripHash}) => {
    const match = /^data:(?<type>.*?),(?<data>.*?)(?:#(?<hash>.*))?$/.exec(urlString);

    if (!match) {
        throw new Error(`Invalid URL: ${urlString}`);
    }

    let {type, data, hash} = match.groups;

    // ...
}
```

✓ `data:png,DEADBEEFCAFE`

✓ `data:jpg,DEADBEEFCAFE#value`

✓ `data:,DEADBEEFCAFE`

OOPS! You’ve Got CVE-2021-33502

```
const normalizeDataURL = (urlString, {stripHash}) => {
    const match = /^data:(?<type>.*?),(?<data>.*?)(?:#(?<hash>.*))?$/.exec(urlString);

    if (!match) {
        throw new Error(`Invalid URL: ${urlString}`);
    }

    let {type, data, hash} = match.groups;

    // ...
}
```

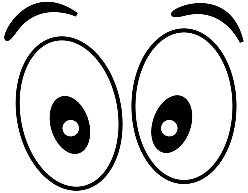
OOPS! You’ve Got `CVE-2021-33502`

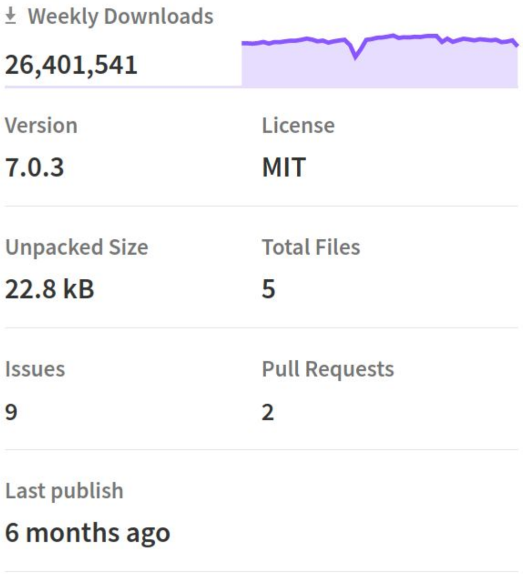
```
const normalizeDataURL = (urlString, {stripHash}) => {
    const match = /^data:(?<type>.*?),(?<data>.*?)(?:#(?<hash>.*))?$/.exec(urlString);

    if (!match) {
        throw new Error(`Invalid URL: ${urlString}`);
    }

    let {type, data, hash} = match.groups;

    // ...
}
```





Weekly Downloads
26,401,541

Version	License
7.0.3	MIT

Unpacked Size	Total Files
22.8 kB	5

Issues	Pull Requests
9	2

Last publish
6 months ago

… but you’re in good company

ReDoS: An Algorithmic Complexity Attack

- Denial-of-Service (DoS)
  - Attacker seeks to deny access to a resource
  - Ideally, attacker seeks amplification to achieve asymmetry
  - Consuming *more* victim resources with *less* effort
  - Disrupting system availability, Denying others fair access to victim resources
- One source of amplification is unintended *complexity* in victim software
  - Remote attackers may leverage this to force worst-case execution
  - Consumes execution time on the victim
- **Regular Expressions (regexps) have complexity!**

Significant Real Impact

> In the course of the study, we identify 25 previously unknown vulnerabilities in popular modules and test 2,846 of the most popular websites against them. We find that 339 of these websites suffer from at least one ReDoS vulnerability.

Freezing the Web: A Study of ReDoS Vulnerabilities in JavaScript-based Web Servers
Cristian-Alexandru Staicu and Michael Pradel, USENIX ‘18

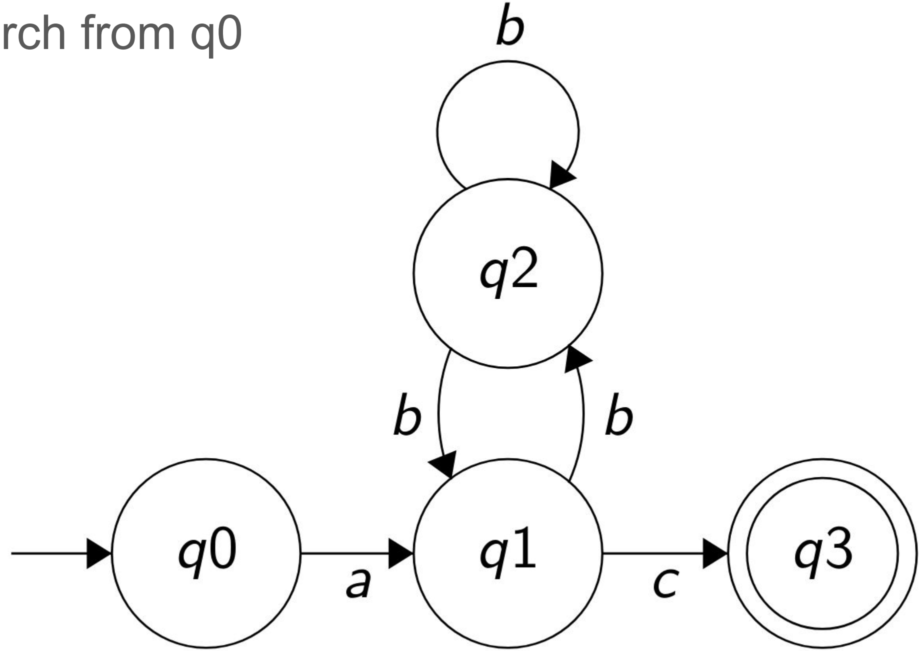
Incremental, But Incomplete Prior Results

- Several prior studies examining ReDoS detectors report high false-positive and high true-positive rates on commonly cited tools
- Shen[1] reports 45% false-negative rate for RXXR2
- Liu[2] reports 97% false-positive rate for Rexploiter, 48% false-negative rate for Shen’s tool ReScue
- All have limitations on feature support

1. *Rescue: Crafting Regular Expression DoS Attacks* Yuju Shen, Yanyan Jiang, Chang Xu, Ping Yu, Xiaoxing Ma, and Jian Lu, ASE ‘18
2. *Revealer: Detecting and Exploiting Regular Expression Denial-of-Service Vulnerabilities* Yinxi Liu, Mingxue Zhang, and Wei Meng, Security & Privacy ‘21

Overview (automaton) / Prior diagnostic attempts

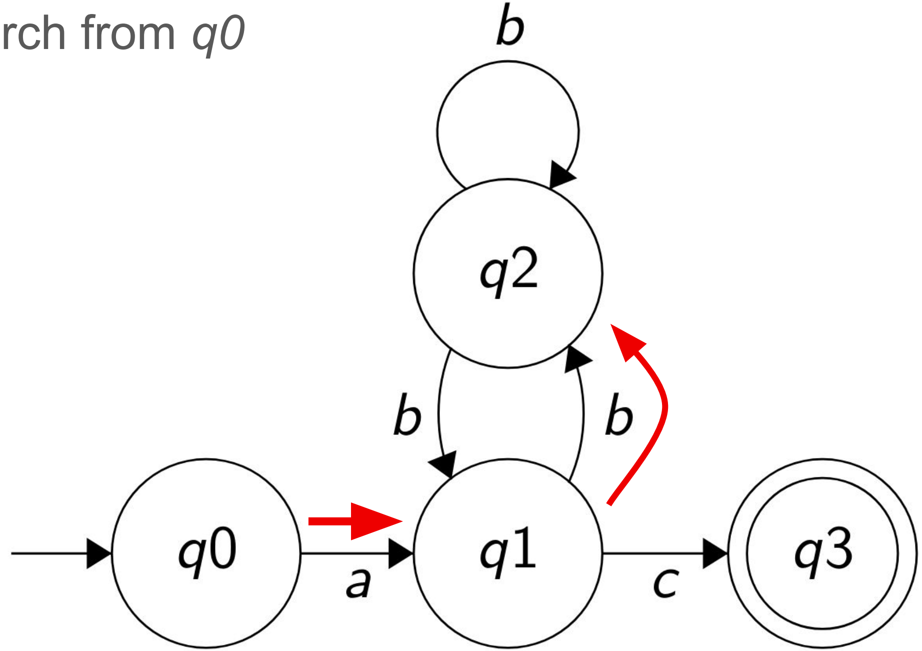
- Notice: Ambiguity in this NFA means we may need to backtrack!
- Attempt constrained depth-first search from q0
- When at *q2*, **where do we go?**



Overview (automaton) / Prior diagnostic attempts

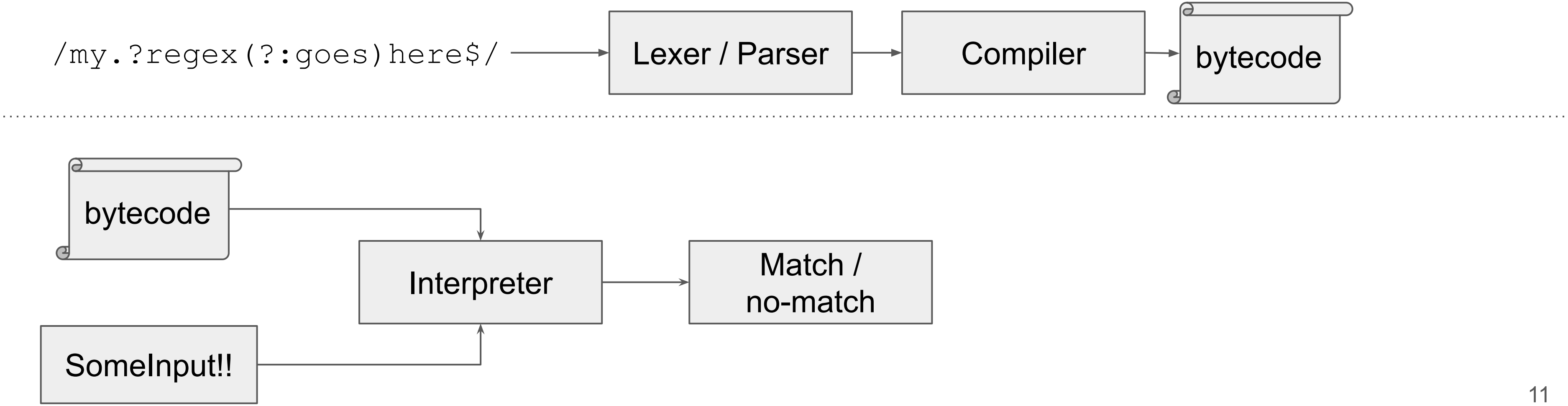
- Notice: Ambiguity in this NFA means we may need to backtrack!
- Attempt constrained depth-first search from *q0*
- When at *q2*, **where do we go?**

abbbbb



Regular Expressions Are Software, Too!

- Much research work has been done on making regular expressions efficient
- BUT this requires long compile times extra space
- Instead, “give up” -- modern regex engines use Spencer’s backtracking alg.
  - Translates a regex into bytecode, as literally as possible



`/my.?regex(?:goes)here$/`

Lexer / Parser

Compiler

bytecode

bytecode

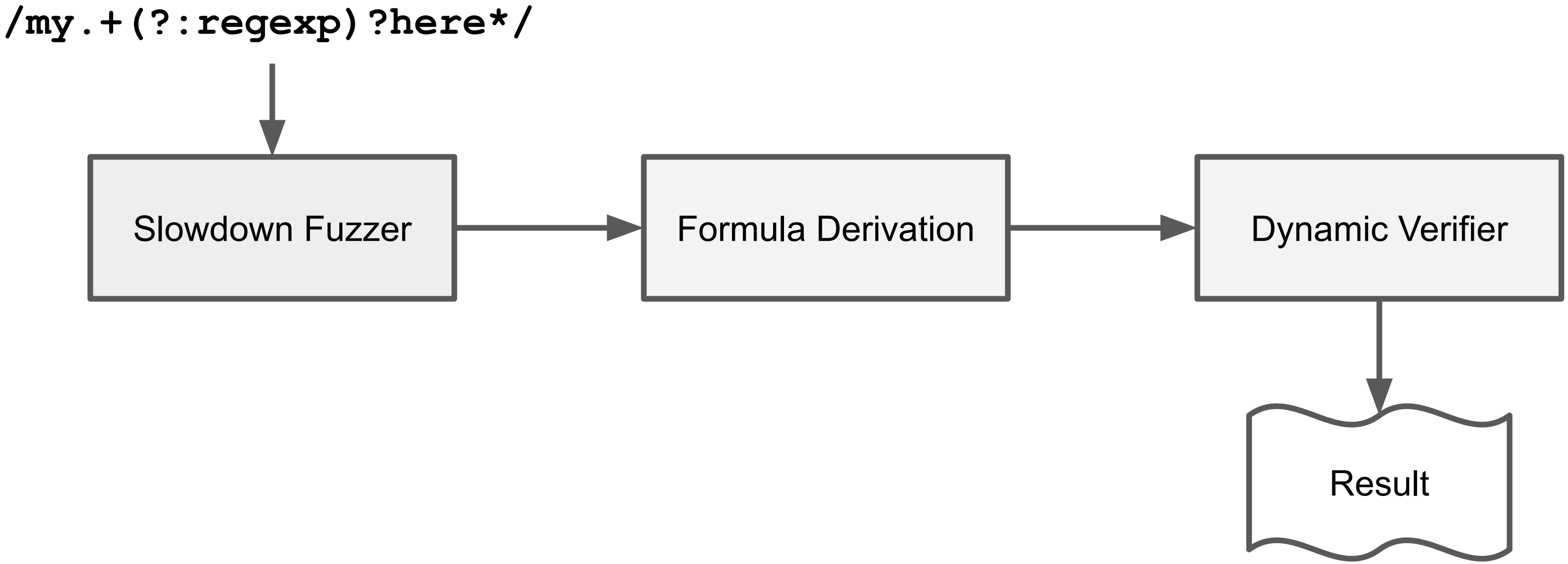
SomeInput!!

Interpreter

Match / no-match

We need a new approach

Regulator: A Simple Solution



`/my.+(?:regexp)?here*/`

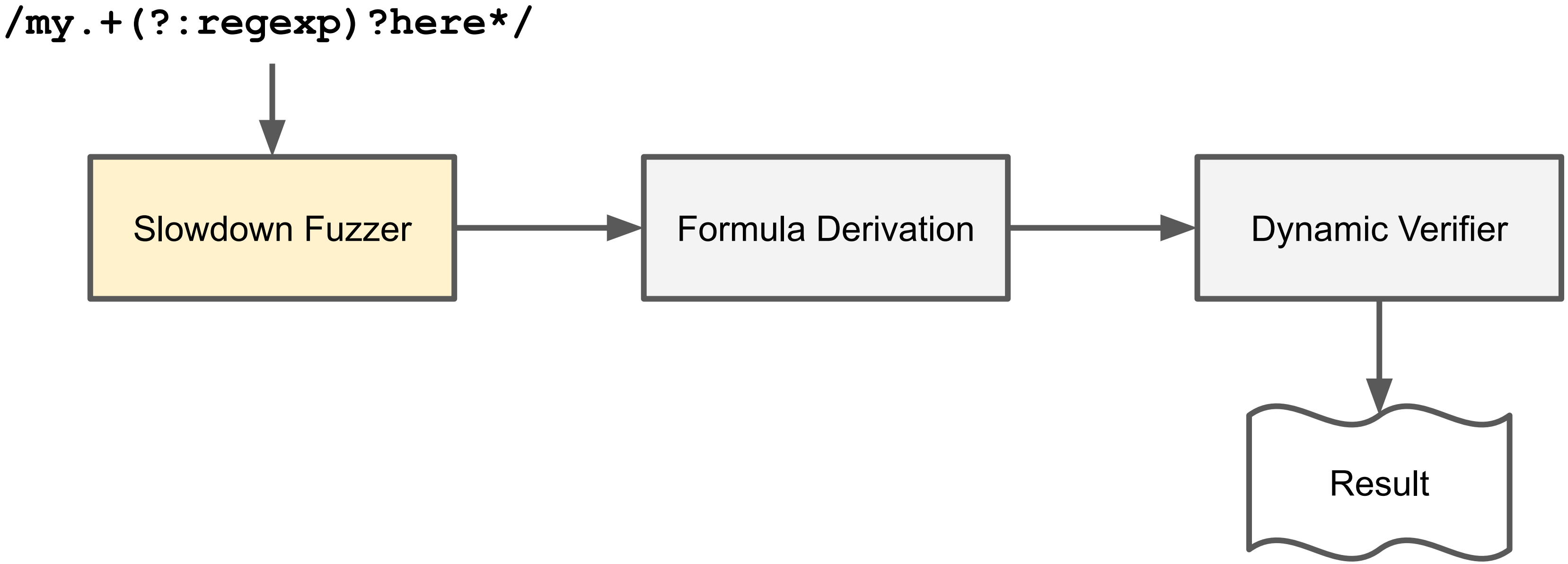
Slowdown Fuzzer

Formula Derivation

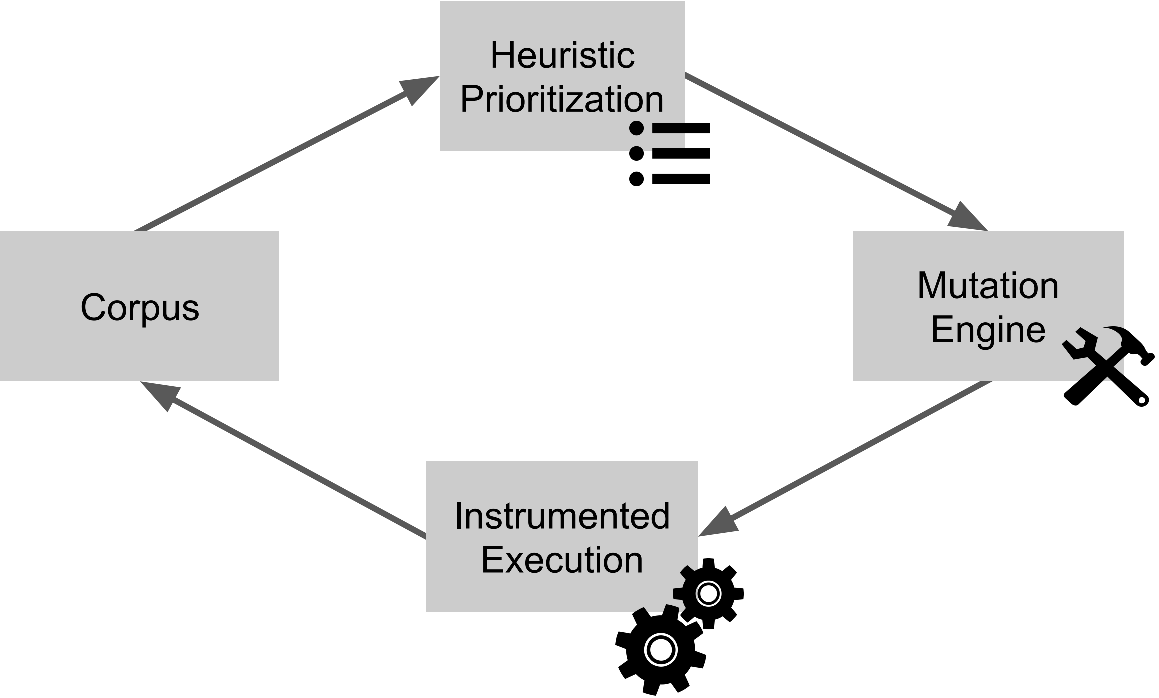
Dynamic Verifier

Result

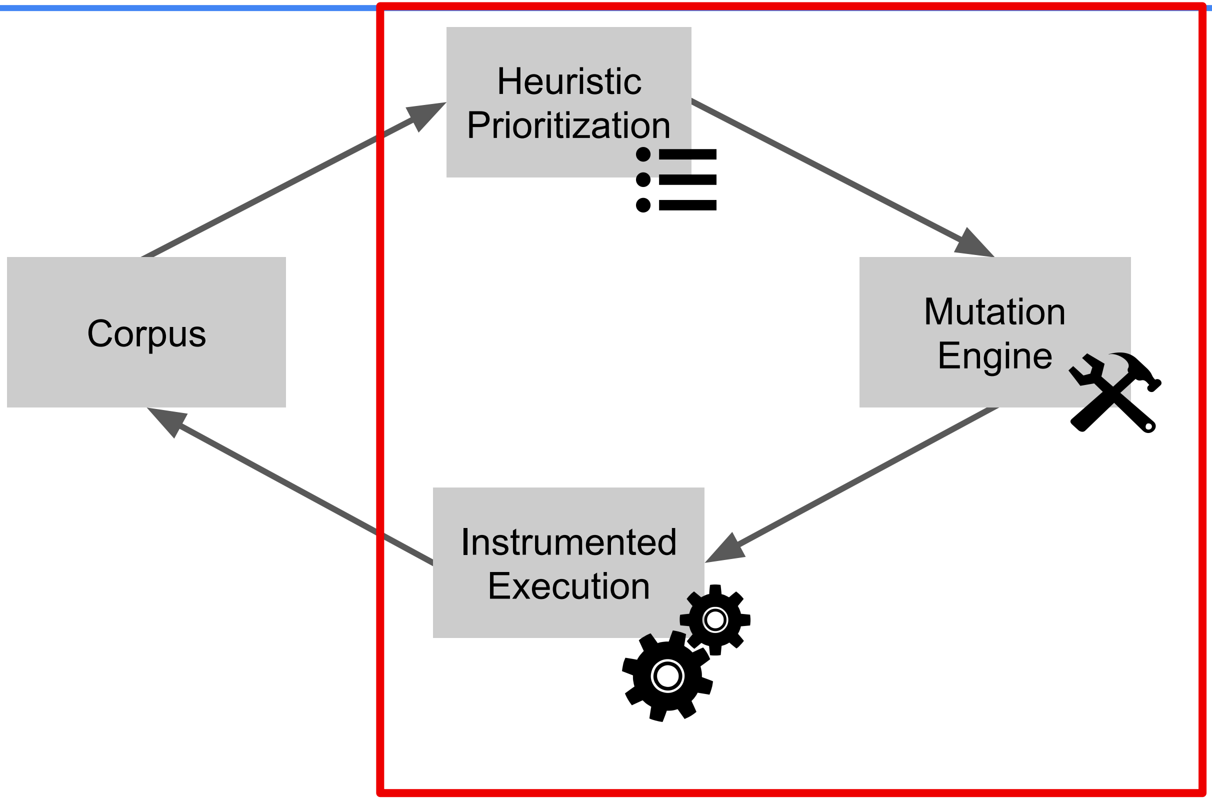
Regulator: A Simple Solution



Regulator / Fuzzer



Regulator / Fuzzer



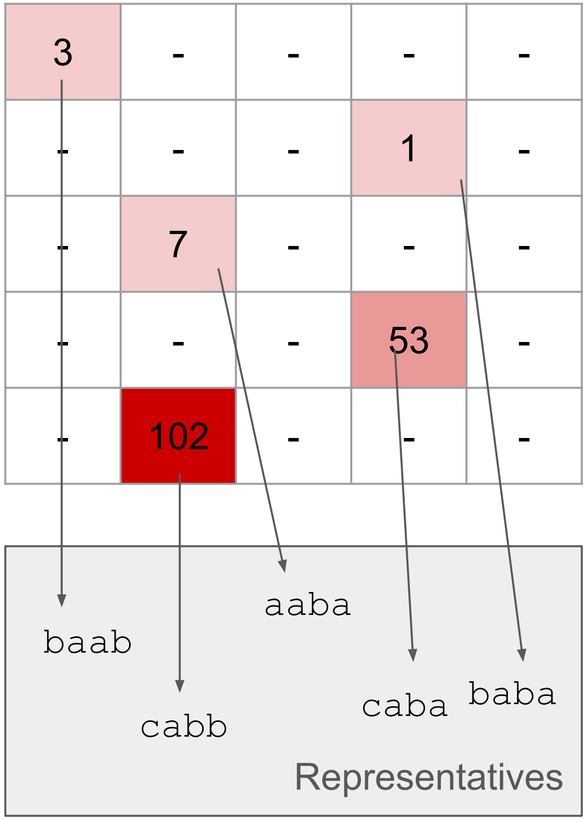
Regulator / Fuzzer / Instrumented Execution

- We instrument the regexp bytecode interpreter itself, `Irregexp`
- AFL-style perfmap to efficiently approximate coverage
  - Each cell counts the hits of a particular branch

3	-	-	-	-
-	-	-	1	-
-	7	-	-	-
-	-	-	53	-
-	102	-	-	-

Regulator / Fuzzer / Heuristic Prioritization

- Problem: We want to prioritize *longest path* while avoiding *local maxima*
- Solution: Prioritize *maximizing representatives* from each perfmap component
- Problem: Only some components are in the *hot path*, prioritizing others is wasted work
- Solution: De-prioritize “stale” inputs with no recent novel offspring inputs



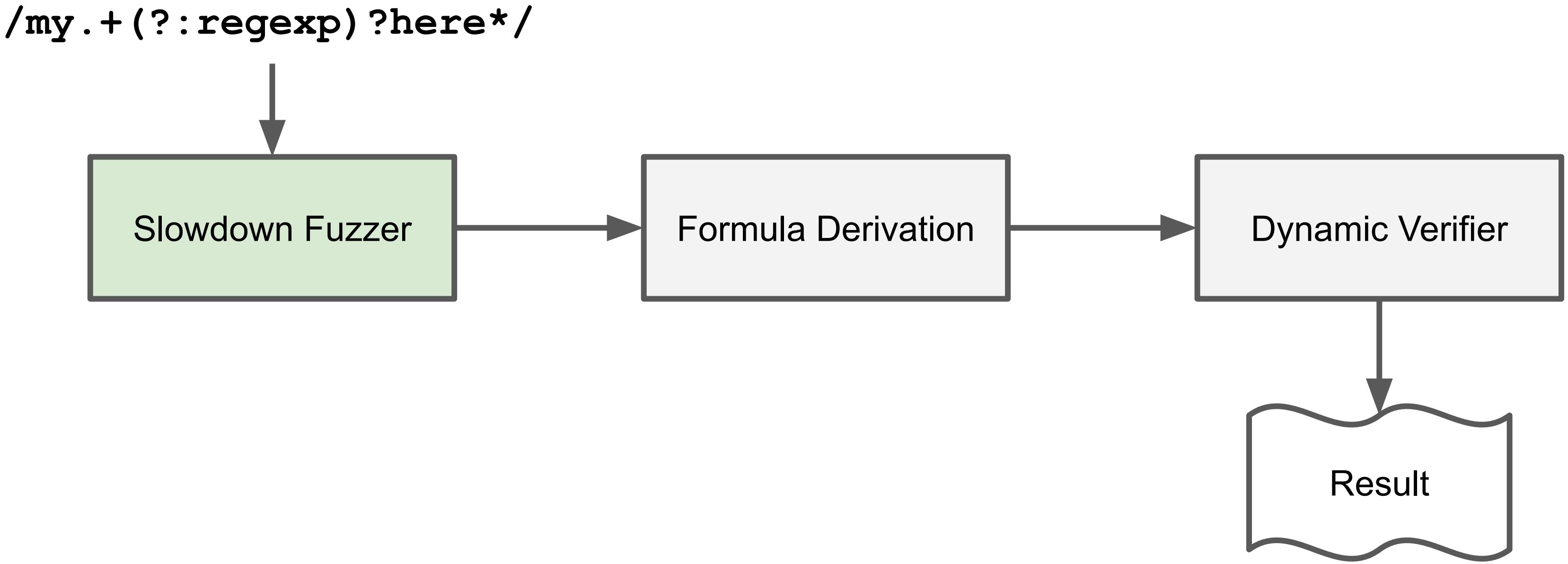
3	-	-	-	-
-	-	-	1	-
-	7	-	-	-
-	-	-	53	-
-	102	-	-	-

Representatives

Regulator / Fuzzer / Mutation Engine “Suggestion”

- The regexp bytecode virtual machine has one *loaded character* register
- We keep a shadow register which remembers *from where in the string* the character was loaded
- On every character-comparison branch, we store
  - (1) the source index of the current loaded character (shadow register)
  - (2) the character(s) which would have negated the branch condition
- Then, during mutation, we will randomly replace the character at that index with one which negates the branch – a “Suggestion”

Regulator: A Simple Solution



`/my.+(?:regexp)?here*/`

Slowdown Fuzzer

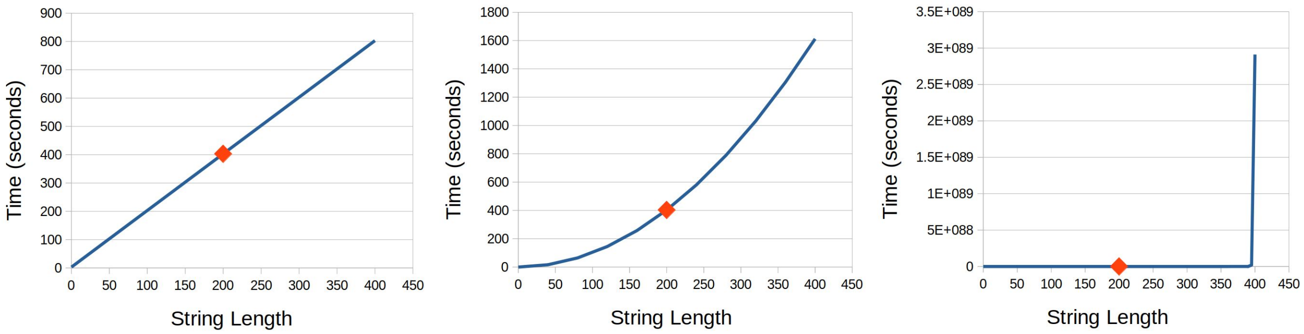
Formula Derivation

Dynamic Verifier

Result

Regulator: ReDoS Formula Derivation

- We now have a costly string (the *witness*)
    - Needs more information! Is this bad? How bad?



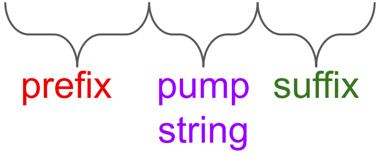
Malign inputs take the form: *attack prefix* + (*pump string*)^n + *attack suffix*

Regulator: ReDoS Formula Derivation

- Observation: with high likelihood, the *witness string* exemplifies the formula
- Approach: we try all (*prefix*, *pump string*, *suffix*) combinations
- When a we find a slow example, we time 20 sample points from 10 to 256 pumps, and fit the result to either linear, power, or exponential regression

Regexp: `hello(1234)*\d*world`

Witness: `hello1234worl`



prefix

pump string

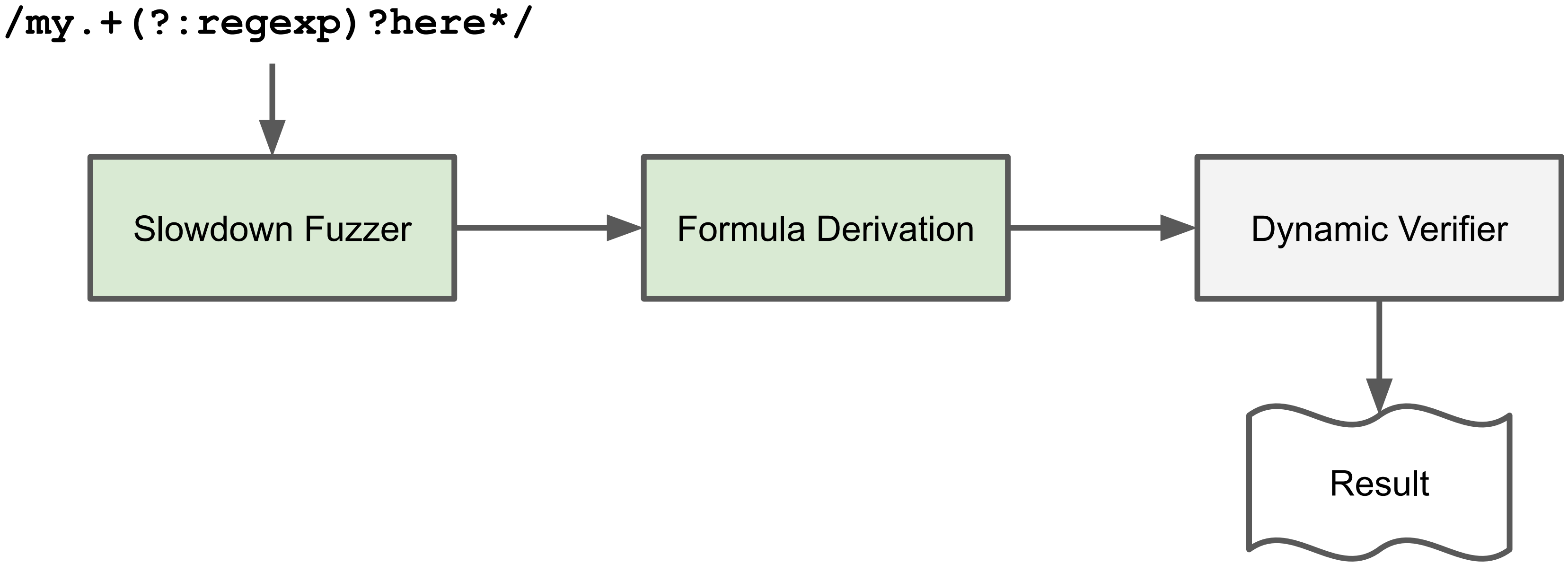
suffix

Example:

```
hello1234worl
hello12341234worl
hello123412341234worl
```

…

Regulator: A Simple Solution



Regulator: Dynamic Verification

- Need to verify correctness of derived ReDoS formula
- Approach:
  - Derive the longest attack string under 1 million characters
  - Following prior work, check for 10 seconds of slow-down[1]
- Once verified, we binary-search for the shortest string with 10s slow-down

[1] *Why aren't regular expressions a lingua franca? An empirical study on the re-use and portability of regular expressions.* James C Davis, Louis G Michael IV, Christy A Coghlan, Francisco Servant, and Dongyoon Lee, ESEC/FSE '19

Evaluation

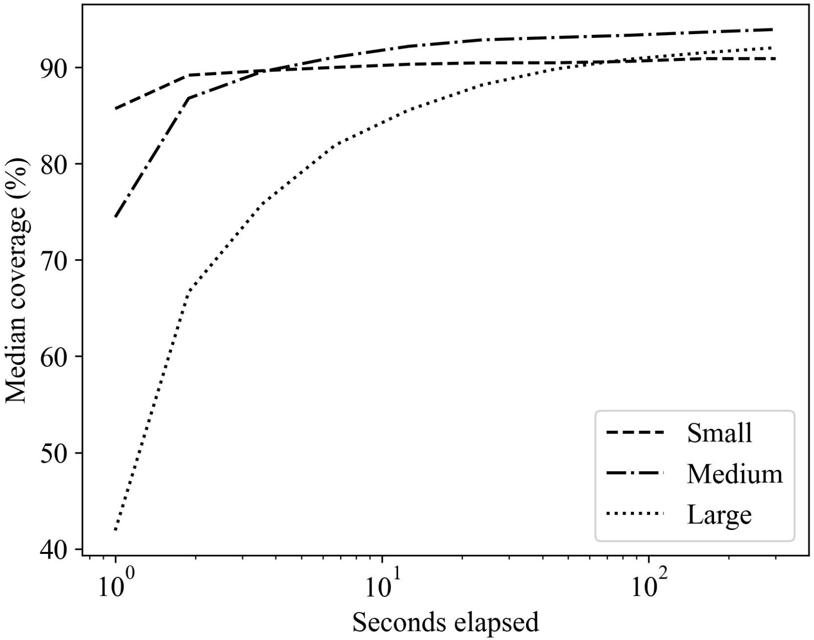
- Against prior work
  - RXXR2
  - Rexploiter
  - NFAA
  - ReScue
  - Revealer
  - PerfFuzz[1] fuzzer + Regulator backend*

- Dataset:
- 3 standard regular expression corpuses:
  - Snort - 13,957 samples
  - RegExLib - 2,990 samples
  - Corpus - 10,037 samples
- Scrape of NPM (package manager for JavaScript), using both Abstract Syntax Tree crawling and basic constant folding
  - 42,743 samples

1. *PerfFuzz: Automatically Generating Pathological Inputs* Caroline Lemieux, Rohan Padhye, Koushik Sen, and Dawn Song. ACM SIGSOFT ‘18

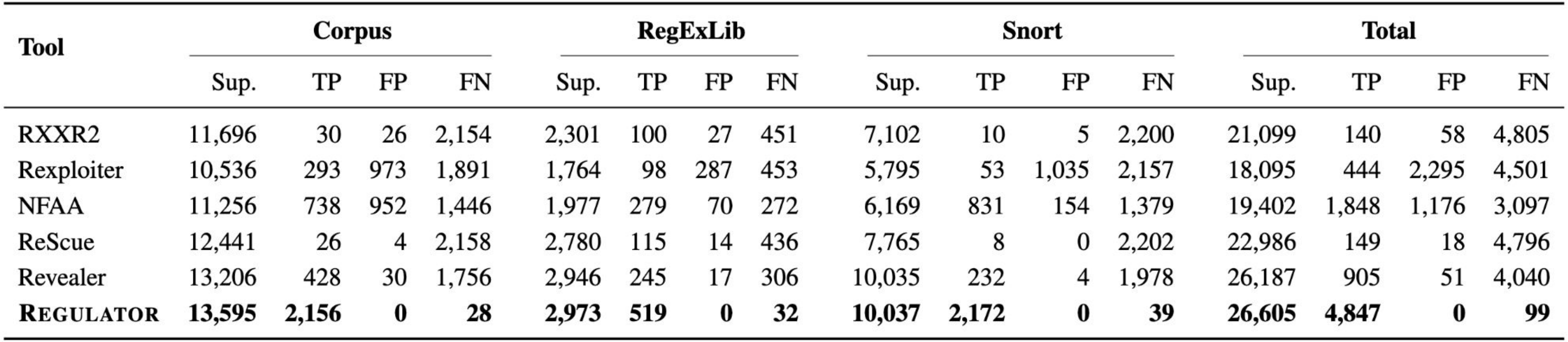
Research Question 1: Is Regulator's Fuzzer Effective?

- We need to ensure high coverage of the regular expression bytecode
- We partition the samples into thirds by bytecode instruction count



Research Question 2: How Does Regulator Compare?

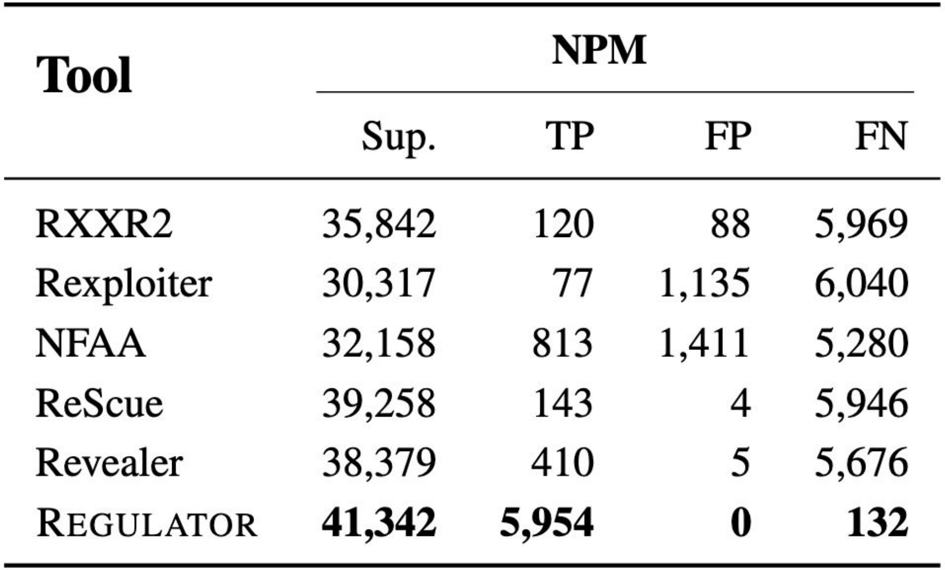
Other ReDoS detection tools (*base* dataset)



Tool	Corpus				RegExLib				Snort				Total			
	Sup.	TP	FP	FN	Sup.	TP	FP	FN	Sup.	TP	FP	FN	Sup.	TP	FP	FN
RXXR2	11,696	30	26	2,154	2,301	100	27	451	7,102	10	5	2,200	21,099	140	58	4,805
Rexploiter	10,536	293	973	1,891	1,764	98	287	453	5,795	53	1,035	2,157	18,095	444	2,295	4,501
NFAA	11,256	738	952	1,446	1,977	279	70	272	6,169	831	154	1,379	19,402	1,848	1,176	3,097
ReScue	12,441	26	4	2,158	2,780	115	14	436	7,765	8	0	2,202	22,986	149	18	4,796
Revealer	13,206	428	30	1,756	2,946	245	17	306	10,035	232	4	1,978	26,187	905	51	4,040
REGULATOR	**13,595**	**2,156**	**0**	**28**	**2,973**	**519**	**0**	**32**	**10,037**	**2,172**	**0**	**39**	**26,605**	**4,847**	**0**	**99**

Research Question 2: How Does Regulator Compare?

Other ReDoS detection tools (*npm* dataset)



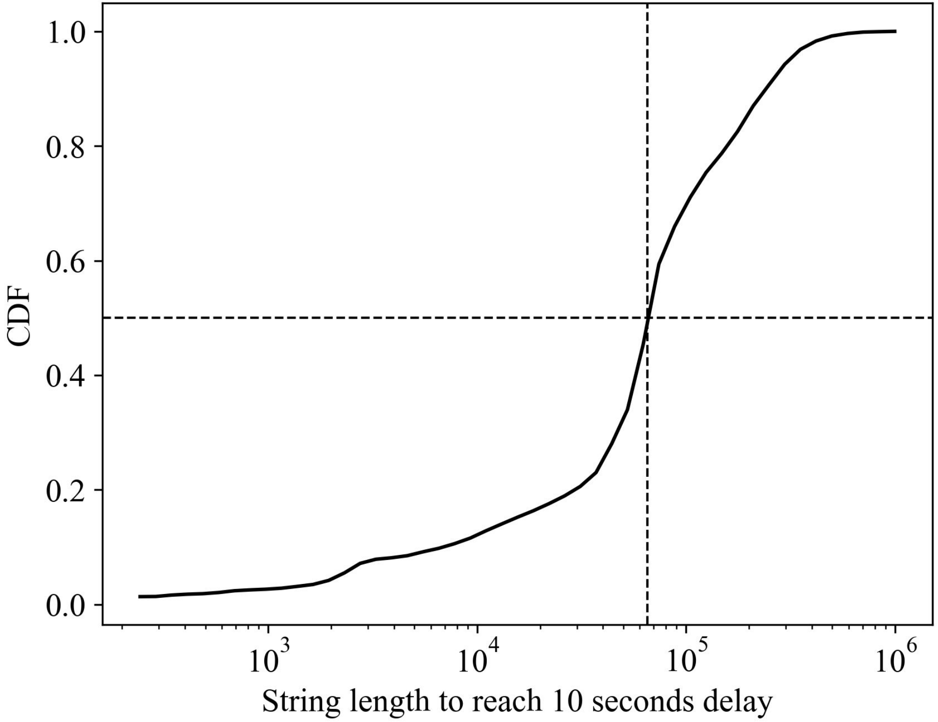
Tool	NPM			
	Sup.	TP	FP	FN
RXXR2	35,842	120	88	5,969
Rexploiter	30,317	77	1,135	6,040
NFAA	32,158	813	1,411	5,280
ReScue	39,258	143	4	5,946
Revealer	38,379	410	5	5,676
REGULATOR	**41,342**	**5,954**	**0**	**132**

Research Question 3: Novel Detections?

- 6 assigned CVE numbers
- Dozens of verified & fixed bugs
- Several difficult-to-find detections
    - Requires certain flags to be set
    - High complexity trips up other analyses
    - Feature use: back-references, look-arounds, special character groups, etc

A Recommendation

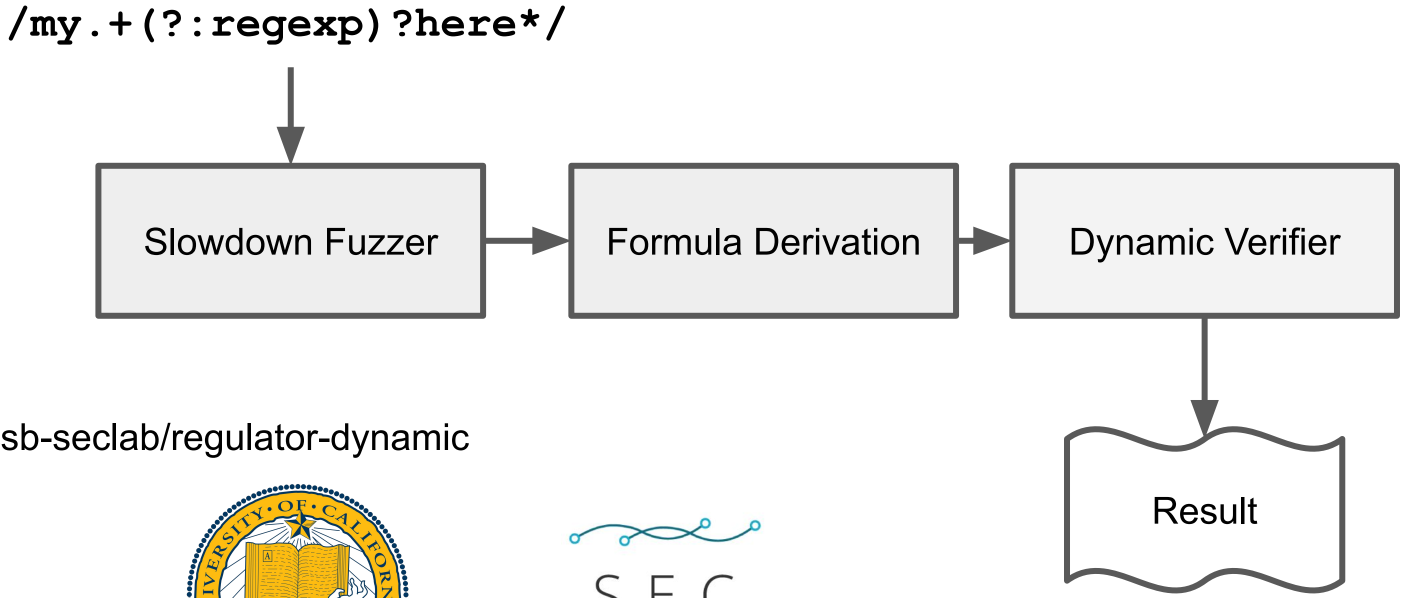
- Limit attacker-controlled string lengths
- A limit of 1,000 or 10,000 characters mitigates nearly all ReDoS that we observed



Thank you!

ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL



https://github.com/ucsb-seclab/regulator-dynamic

@robmcl4

robert349@ucsb.edu

SEC LAB